Full-Stack Application Deployment - Standard Operating Procedure (SOP)

Version: 1.0
Last Updated: January 21, 2026
Author: Wizone IT Solutions
Purpose: Step-by-step guide for deploying full-stack applications with Docker, CI/CD, and SSL

▯ Table of Contents

▯ Overview

This SOP covers the complete deployment process for full-stack applications using:

- **Frontend:** React/Vue/Angular with Nginx
- **Backend:** Node.js/Express API
- **Database:** Microsoft SQL Server / PostgreSQL / MySQL
- **Containerization:** Docker & Docker Compose
- **CI/CD:** GitHub Actions
- **Reverse Proxy:** Nginx with SSL
- **SSL Certificates:** Let's Encrypt (free)

Deployment Architecture:

```
Internet → Domain (SSL) → Nginx Reverse Proxy → Docker Containers
                                                  ├─ Frontend
                                                  ├─ Backend API
                                                  └─ Database
```

▯ Prerequisites

Required Access & Credentials

- ☐ VPS server (Ubuntu 20.04+ or Debian 11+)
- ☐ Root or sudo user access
- ☐ Domain name (purchased and accessible)
- ☐ GitHub account with repository access
- ☐ Docker Hub account (for image hosting)

Required Software on VPS

- ☐ Docker (version 20.10+)
- ☐ Docker Compose (version 1.29+)
- ☐ Git
- ☐ Nginx
- ☐ Certbot (for SSL)
- ☐ SSH access (port 22 or custom)

Local Development Machine

- [ ] Git installed
- [ ] Code editor (VS Code recommended)
- [ ] SSH client
- [ ] Basic knowledge of Linux commands

□ Project Preparation

Step 1: Project Structure

Ensure your project follows this structure:

```
project-root/
├── .github/
│   └── workflows/
│       └── deploy.yml          # CI/CD pipeline
├── .gitignore                  # Root gitignore
├── README.md                   # Project documentation
├── docker-compose.yml          # Production compose file
├── docker-compose.dev.yml      # Development compose file (optional)
│
├── backend/
│   ├── Dockerfile             # Backend Docker configuration
│   ├── .dockerignore          # Docker ignore rules
│   ├── .gitignore             # Backend gitignore
│   ├── package.json
│   ├── tsconfig.json          # If using TypeScript
│   └── src/
│       └── (source code)
│
└── frontend/
    ├── Dockerfile             # Frontend Docker configuration
    ├── nginx.conf             # Nginx configuration for SPA
    ├── .dockerignore          # Docker ignore rules
    ├── .gitignore             # Frontend gitignore
    ├── package.json
    └── src/
        └── (source code)
```

Step 2: Create Essential Files

2.1 Backend Dockerfile

```
 # backend/Dockerfile
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
RUN npm run build

FROM node:20-alpine
WORKDIR /app
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/package*.json ./
EXPOSE 8080
CMD ["node", "dist/server.js"]
```

2.2 Frontend Dockerfile

```
# frontend/Dockerfile
FROM node:20-alpine AS builder
WORKDIR /app
ARG VITE_API_BASE_URL
ENV VITE_API_BASE_URL=${VITE_API_BASE_URL}
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80 443
CMD ["nginx", "-g", "daemon off;"]
```

2.3 Frontend nginx.conf

```
# frontend/nginx.conf
server {
    listen 80;
    listen 443 ssl;
    server_name _;
    root /usr/share/nginx/html;
    index index.html;

    # Gzip compression
    gzip on;
    gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascr

    # SPA fallback
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Cache static assets
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg|woff|woff2|ttf|eot)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }
}
```

2.4 docker-compose.yml

```yaml
version: '3.8'

services:
  backend:
    image: ${DOCKER_USERNAME}/project-name:backend
    container_name: project-backend
    restart: unless-stopped
    ports:
      - "8080:8080"
    env_file:
      - .env
    depends_on:
      - database
    networks:
      - app-network
    healthcheck:
      test: ["CMD", "wget", "--no-verbose", "--tries=1", "--spider", "http://localhost:8080/api/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  frontend:
    image: ${DOCKER_USERNAME}/project-name:frontend
    container_name: project-frontend
    restart: unless-stopped
    ports:
      - "8081:80"
      - "8443:443"
    networks:
      - app-network
    healthcheck:
      test: ["CMD", "wget", "--no-verbose", "--tries=1", "--spider", "http://localhost"]
      interval: 30s
      timeout: 10s
      retries: 3

  database:
    image: mcr.microsoft.com/mssql/server:2022-latest  # or postgres:15-alpine, mysql:8
    container_name: project-database
    restart: unless-stopped
    environment:
      - ACCEPT_EULA=Y
      - SA_PASSWORD=${DB_PASSWORD}
      - MSSQL_PID=Express
    ports:
      - "1433:1433"
    volumes:
      - db-data:/var/opt/mssql
    networks:
      - app-network
    healthcheck:
      test: ["CMD", "/opt/mssql-tools18/bin/sqlcmd", "-S", "localhost", "-U", "sa", "-P", "${DB_PASSWORD}", "-Q", "SELECT 1", "-C"]
      interval: 30s
      timeout: 10s
      retries: 5

volumes:
  db-data:
    driver: local

networks:
  app-network:
    driver: bridge
```

2.5 .gitignore (Root)

```
# Environment files
.env
.env.local
.env.production
.env.*.local
*.env

# IDE
.vscode/
.idea/
*.swp

# OS
.DS_Store
Thumbs.db

# Logs
*.log
logs/

# Dependencies
node_modules/

# Build
dist/
build/

# Secrets
*.pem
*.key
secrets/
```

Step 3: Environment Variables Template

Create `.env.example` file:

```
# Database Configuration
DATABASE_URL=sqlserver://database:1433;database=app_db;user=sa;password=CHANGE_ME;encrypt=true;trustServerCertificate=true
DB_PASSWORD=CHANGE_ME

# JWT Configuration
JWT_SECRET=CHANGE_ME_RANDOM_STRING_64_CHARS
JWT_REFRESH_SECRET=CHANGE_ME_ANOTHER_RANDOM_STRING
JWT_EXPIRES_IN=7d
JWT_REFRESH_EXPIRES_IN=30d

# Server Configuration
NODE_ENV=production
PORT=8080

# CORS Configuration
CORS_ORIGIN=https://yourdomain.com

# Frontend Configuration
VITE_API_BASE_URL=https://yourdomain.com/api

# Docker Hub
DOCKER_USERNAME=your-dockerhub-username
```

� VPS Server Setup

Step 1: Initial Server Access

```
# Connect to VPS via SSH
ssh root@YOUR_VPS_IP

# Or with custom port
ssh -p 2211 username@YOUR_VPS_IP
```

Step 2: Create Deployment User

```
# Create deployment user
adduser deployer

# Add to sudo group
usermod -aG sudo deployer

# Set up SSH for new user
mkdir -p /home/deployer/.ssh
cp ~/.ssh/authorized_keys /home/deployer/.ssh/
chown -R deployer:deployer /home/deployer/.ssh
chmod 700 /home/deployer/.ssh
chmod 600 /home/deployer/.ssh/authorized_keys

# Switch to deployment user
su - deployer
```

Step 3: Install Required Software

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER
newgrp docker

# Verify Docker installation
docker --version
docker run hello-world

# Install Docker Compose
sudo curl -L "https://github.com/docker/compose/releases/download/v2.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version

# Install Nginx
sudo apt install nginx -y
sudo systemctl enable nginx
sudo systemctl start nginx

# Install Certbot (for SSL)
sudo apt install certbot python3-certbot-nginx -y

# Install Git
sudo apt install git -y
git --version
```

Step 4: Configure Firewall

```
# Enable UFW firewall
sudo ufw enable

# Allow SSH (adjust port if custom)
sudo ufw allow 22/tcp
# Or for custom port: sudo ufw allow 2211/tcp

# Allow HTTP and HTTPS
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp

# Check status
sudo ufw status
```

Step 5: Create Project Directory

```
# Create application directory
sudo mkdir -p /var/www/htdocs/project-name
sudo chown -R $USER:$USER /var/www/htdocs/project-name
cd /var/www/htdocs/project-name
```

Domain & SSL Configuration

Step 1: DNS Configuration

In your domain registrar (GoDaddy, Namecheap, etc.):

1. Log in to domain management panel
2. Go to DNS settings
3. Add/Update A records:

```
Type: A
Name: @ (or root domain)
Value: YOUR_VPS_IP_ADDRESS
TTL: 3600 (or default)

Type: A
Name: www
Value: YOUR_VPS_IP_ADDRESS
TTL: 3600
```

4. Wait 5-30 minutes for DNS propagation
5. Verify DNS: `nslookup yourdomain.com`

Step 2: Nginx Reverse Proxy Configuration

```
# Create Nginx configuration
sudo nano /etc/nginx/sites-available/project-name
```

Nginx Configuration File:

```
# HTTP - Redirect to HTTPS (will be configured after SSL)
server {
    listen 80;
    server_name yourdomain.com www.yourdomain.com;

    # For SSL certificate verification
    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    # Temporary: Allow access before SSL
    location / {
        proxy_pass http://localhost:8081;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

```
# Enable site
sudo ln -s /etc/nginx/sites-available/project-name /etc/nginx/sites-enabled/

# Test Nginx configuration
sudo nginx -t

# Reload Nginx
sudo systemctl reload nginx
```

Step 3: Obtain SSL Certificate

```
# Create certbot directory
sudo mkdir -p /var/www/certbot

# Obtain SSL certificate
sudo certbot --nginx -d yourdomain.com -d www.yourdomain.com

# Follow prompts:
# - Enter email address
# - Agree to terms
# - Choose redirect HTTP to HTTPS: Yes (2)

# Verify auto-renewal
sudo certbot renew --dry-run
```

Step 4: Update Nginx Configuration (Post-SSL)

```
sudo nano /etc/nginx/sites-available/project-name
```

Complete Nginx Configuration with SSL:

```
# HTTP - Redirect to HTTPS
server {
    listen 80;
    server_name yourdomain.com www.yourdomain.com;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$server_name$request_uri;
    }
}

# HTTPS - Main Application
server {
    listen 443 ssl http2;
    server_name yourdomain.com www.yourdomain.com;

    # SSL Configuration
    ssl_certificate /etc/letsencrypt/live/yourdomain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/yourdomain.com/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # Security Headers
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;

    # Frontend (React/Vue/Angular SPA)
    location / {
        proxy_pass http://localhost:8081;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Backend API
    location /api {
        proxy_pass http://localhost:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Increase timeouts for long-running requests
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;
    }

    # WebSocket support (if needed)
    location /ws {
        proxy_pass http://localhost:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_set_header Host $host;
    }
}
```

```
# Test and reload
sudo nginx -t
sudo systemctl reload nginx
```

󰊢 Docker Configuration

Step 1: Login to Docker Hub

```
# On VPS
docker login

# Enter Docker Hub username and password/token
# Username: your-dockerhub-username
# Password: your-docker-hub-token
```

Step 2: Create .env File

```
cd /var/www/htdocs/project-name

# Create .env file
nano .env
```

Paste your production environment variables:

```
DATABASE_URL=sqlserver://database:1433;database=app_db;user=sa;password=STRONG_PASSWORD_HERE;encrypt=true;trustServerCertificate=tru
DB_PASSWORD=STRONG_PASSWORD_HERE
JWT_SECRET=RANDOM_64_CHAR_STRING_GENERATE_WITH_OPENSSL
JWT_REFRESH_SECRET=ANOTHER_RANDOM_64_CHAR_STRING
JWT_EXPIRES_IN=7d
JWT_REFRESH_EXPIRES_IN=30d
NODE_ENV=production
PORT=8080
CORS_ORIGIN=https://yourdomain.com
VITE_API_BASE_URL=https://yourdomain.com/api
DOCKER_USERNAME=your-dockerhub-username
```

Generate Secure Secrets:

```
# Generate JWT secrets
openssl rand -base64 64

# Generate strong database password
openssl rand -base64 32
```

Step 3: Clone Repository

```
cd /var/www/htdocs/project-name

# Clone repository
git clone https://github.com/your-username/project-name.git .

# Or if already cloned, pull latest
git pull origin main
```

󰊢 GitHub Repository Setup

Step 1: Create GitHub Repository

1. Go to https://github.com/new
2. Repository name: `project-name`
3. Visibility: Private (recommended) or Public
4. Initialize with README: No (if pushing existing code)
5. Click "Create repository"

Step 2: Push Code to GitHub

```
# On local machine
cd /path/to/your/project

# Initialize git (if not already)
git init

# Add remote
git remote add origin https://github.com/your-username/project-name.git

# Add all files
git add .

# Commit
git commit -m "Initial commit"

# Push to main branch
git branch -M main
git push -u origin main
```

Step 3: Create GitHub Secrets

Navigate to: Repository → Settings → Secrets and variables → Actions → New repository secret

Add these secrets:

Secret Name	Value	Description
`SSH_HOST`	`YOUR_VPS_IP`	VPS IP address
`SSH_USER`	`deployer`	SSH username
`SSH_KEY`	`-----BEGIN...`	Private SSH key (see below)
`DOCKER_USERNAME`	`your-dockerhub`	Docker Hub username
`DOCKER_PASSWORD`	`dckr_pat_xxx`	Docker Hub access token
`DATABASE_URL`	`sqlserver://...`	Production database URL
`DB_PASSWORD`	`strong-password`	Database SA password
`JWT_SECRET`	`random-64-chars`	JWT secret key
`JWT_REFRESH_SECRET`	`random-64-chars`	JWT refresh secret
`CORS_ORIGIN`	`https://domain.com`	Frontend URL
`VITE_API_BASE_URL`	`https://domain.com/api`	Backend API URL

Generate SSH Key for GitHub Actions:

```
# On local machine
ssh-keygen -t rsa -b 4096 -C "github-actions" -f ~/.ssh/github_actions_deploy

# Copy public key to VPS
ssh-copy-id -i ~/.ssh/github_actions_deploy.pub deployer@YOUR_VPS_IP

# Or manually:
# 1. Copy content of github_actions_deploy.pub
cat ~/.ssh/github_actions_deploy.pub

# 2. On VPS, add to authorized_keys
# ssh deployer@YOUR_VPS_IP
# nano ~/.ssh/authorized_keys
# Paste public key and save

# Copy private key content (this goes in GitHub Secret)
cat ~/.ssh/github_actions_deploy
# Copy entire output including -----BEGIN and -----END lines
```

Create Docker Hub Access Token:

1. Go to https://hub.docker.com/settings/security
2. Click "New Access Token"
3. Description: "GitHub Actions"
4. Access permissions: "Read, Write, Delete"
5. Click "Generate"
6. Copy token (starts with `dckr_pat_`)
7. Save this as `DOCKER_PASSWORD` secret in GitHub

⚙ CI/CD Pipeline Setup

Step 1: Create GitHub Actions Workflow

```
# On local machine
mkdir -p .github/workflows
nano .github/workflows/deploy.yml
```

Complete CI/CD Workflow:

```
name: Deploy Application

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

env:
  REGISTRY: docker.io
  IMAGE_NAME: ${{ secrets.DOCKER_USERNAME }}/project-name

jobs:
  test-backend:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./backend
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
```

```
        uses: actions/setup-node@v4
        with:
          node-version: '20'
          cache: 'npm'
          cache-dependency-path: ./backend/package-lock.json

      - name: Install dependencies
        run: npm ci

      - name: Type check
        run: npm run typecheck || echo "Skipping typecheck"

      - name: Lint
        run: npm run lint || echo "Skipping lint"

  test-frontend:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./frontend
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '20'
          cache: 'npm'
          cache-dependency-path: ./frontend/package-lock.json

      - name: Install dependencies
        run: npm ci

      - name: Build
        run: npm run build
        env:
          VITE_API_BASE_URL: ${{ secrets.VITE_API_BASE_URL }}

  build-and-push:
    runs-on: ubuntu-latest
    needs: [test-backend, test-frontend]
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build and Push Backend
        uses: docker/build-push-action@v5
        with:
          context: ./backend
          file: ./backend/Dockerfile
          push: true
          tags: |
            ${{ env.IMAGE_NAME }}:backend
            ${{ env.IMAGE_NAME }}:backend-${{ github.sha }}
          cache-from: type=gha
          cache-from: type=gha
          cache-to: type=gha,mode=max

      - name: Build and Push Frontend
        uses: docker/build-push-action@v5
        with:
          context: ./frontend
          file: ./frontend/Dockerfile
          push: true
          build-args: |
            VITE_API_BASE_URL=${{ secrets.VITE_API_BASE_URL }}
          tags: |
            ${{ env.IMAGE_NAME }}:frontend
            ${{ env.IMAGE_NAME }}:frontend-${{ github.sha }}
          cache-from: type=gha
          cache-to: type=gha,mode=max

  deploy-to-vps:
    needs: build-and-push
    runs-on: ubuntu-latest
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'
    steps:
      - name: Deploy to VPS via SSH
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ secrets.SSH_HOST }}
          username: ${{ secrets.SSH_USER }}
          key: ${{ secrets.SSH_KEY }}
          port: 22  # Change if using custom SSH port
          script: |
            set -e

            # Navigate to project directory
            cd /var/www/htdocs/project-name || exit 1

            # Pull latest code
            git pull origin main || (git clone https://github.com/${{ github.repository }}.git . && git checkout main)

            # Login to Docker Hub
            echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u ${{ secrets.DOCKER_USERNAME }} --password-stdin

            # Create/update .env file
            cat > .env << 'EOF'
            DATABASE_URL=${{ secrets.DATABASE_URL }}
            DB_PASSWORD=${{ secrets.DB_PASSWORD }}
            JWT_SECRET=${{ secrets.JWT_SECRET }}
            JWT_REFRESH_SECRET=${{ secrets.JWT_REFRESH_SECRET }}
            JWT_EXPIRES_IN=7d
            JWT_REFRESH_EXPIRES_IN=30d
            NODE_ENV=production
            PORT=8080
            CORS_ORIGIN=${{ secrets.CORS_ORIGIN }}
            VITE_API_BASE_URL=${{ secrets.VITE_API_BASE_URL }}
            DOCKER_USERNAME=${{ secrets.DOCKER_USERNAME }}
            EOF

            # Stop existing containers
            docker-compose down || true

            # Pull latest images
            docker-compose pull backend frontend

            # Start all services
            docker-compose up -d

            # Wait for services to be healthy
            echo "Waiting for services to start..."
            sleep 30

            # Run database migrations (if applicable)
            docker-compose exec -T backend npm run migrate || echo "Migration skipped or failed"

            # Check container status
            echo "=== Container Status ==="
            docker-compose ps

            # Show recent logs
            echo "=== Backend Logs ==="
            docker-compose logs --tail=20 backend

            echo "=== Frontend Logs ==="
            docker-compose logs --tail=20 frontend

            # Cleanup old images
            docker image prune -af --filter "until=48h" || true

            echo "=== Deployment Complete ==="
```

Step 2: Commit and Push Workflow

```
git add .github/workflows/deploy.yml
git commit -m "ci: add GitHub Actions deployment workflow"
git push origin main
```

Step 3: Monitor Deployment

1. Go to https://github.com/your-username/project-name/actions
2. Click on the latest workflow run
3. Monitor each job:
    - test-backend
    - test-frontend
    - build-and-push
    - deploy-to-vps

Database Configuration

Step 1: Database Initialization

```
# On VPS - Wait for database container to be healthy
cd /var/www/htdocs/project-name
docker-compose ps

# Check database logs
docker-compose logs database

# For MS SQL Server - Verify database
docker-compose exec database /opt/mssql-tools18/bin/sqlcmd -S localhost -U sa -P "${DB_PASSWORD}" -C -Q "SELECT name FROM sys.databa
```

Step 2: Run Migrations

For Prisma (Node.js):

```
# Generate Prisma client
docker-compose exec backend npx prisma generate

# Run migrations
docker-compose exec backend npx prisma migrate deploy

# Seed database (optional)
docker-compose exec backend npx prisma db seed
```

For Sequelize:

```
docker-compose exec backend npx sequelize-cli db:migrate
docker-compose exec backend npx sequelize-cli db:seed:all
```

For TypeORM:

```
docker-compose exec backend npm run typeorm migration:run
docker-compose exec backend npm run seed
```

Step 3: Create Admin User

```
# Example for creating admin user via script
docker-compose exec backend node -e "
const { PrismaClient } = require('@prisma/client');
const bcrypt = require('bcryptjs');

const prisma = new PrismaClient();

async function createAdmin() {
  const hashedPassword = await bcrypt.hash('Admin@123', 10);

  const admin = await prisma.user.upsert({
    where: {
      email_tenantId: {
        email: 'admin@yourdomain.com',
        tenantId: 'default'
      }
    },
    update: {},
    create: {
      email: 'admin@yourdomain.com',
      password: hashedPassword,
      role: 'ADMIN',
      tenantId: 'default',
      companyName: 'Default Company'
    }
  });

  console.log('Admin created:', admin.email);
  process.exit(0);
}

createAdmin().catch(console.error);
"
```

▯ Initial Deployment

Step 1: Manual First Deployment

```
# On VPS
cd /var/www/htdocs/project-name

# Ensure .env file exists with correct values
cat .env

# Pull Docker images
docker-compose pull

# Start services
docker-compose up -d

# Wait for containers to be healthy
sleep 30

# Check status
docker-compose ps

# View logs
docker-compose logs -f
```

Step 2: Verify Services

```
# Check backend health
curl http://localhost:8080/api/health

# Check frontend
curl http://localhost:8081

# Check database connection
docker-compose exec backend node -e "
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();
prisma.\$connect()
  .then(() => { console.log('✓ Database connected'); process.exit(0); })
  .catch(e => { console.error('✗ Connection failed:', e.message); process.exit(1); });
"
```

Step 3: External Access Test

```
# Test HTTPS access
curl -k https://yourdomain.com

# Test API
curl -k https://yourdomain.com/api/health

# Test login endpoint
curl -X POST https://yourdomain.com/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"admin@yourdomain.com","password":"Admin@123"}'
```

✅ Verification & Testing

Checklist

- [ ] **DNS Resolution:** `nslookup yourdomain.com` returns correct IP
- [ ] **SSL Certificate:** https://yourdomain.com shows padlock icon
- [ ] **Frontend Loading:** Application loads without errors
- [ ] **Backend API:** https://yourdomain.com/api/health returns 200
- [ ] **Database Connection:** Backend can connect to database

- [ ] **Login Function:** Users can login successfully
- [ ] **CORS Configuration:** No CORS errors in browser console
- [ ] **Container Health:** All containers show "healthy" status
- [ ] **Logs Review:** No critical errors in logs
- [ ] **Auto-Restart:** Containers restart after VPS reboot

Testing Commands

```
# Health checks
docker-compose exec backend wget --spider http://localhost:8080/api/health
docker-compose exec frontend wget --spider http://localhost

# Container stats
docker stats

# Resource usage
docker-compose exec backend ps aux
docker-compose exec backend df -h

# Network connectivity
docker-compose exec backend ping -c 3 database
docker-compose exec frontend ping -c 3 backend
```

Browser Testing

1. **Frontend Access:** https://yourdomain.com
2. **API Documentation:** https://yourdomain.com/api/docs (if Swagger enabled)
3. **Login Test:** Use admin credentials
4. **Network Tab:** Check for 200 responses, no CORS errors
5. **Console Tab:** Check for no JavaScript errors
6. **Application Tab:** Verify JWT token storage

🔧 Troubleshooting

Common Issues & Solutions

Issue 1: Container Won't Start

Symptoms:

- Container status shows "Restarting" or "Exited"
- `docker-compose ps` shows unhealthy containers

Solutions:

```
# Check logs
docker-compose logs backend
docker-compose logs frontend
docker-compose logs database

# Check container inspect
docker inspect <container-name>

# Restart containers
docker-compose restart

# Rebuild containers
docker-compose up -d --force-recreate
```

Issue 2: Database Connection Timeout

Symptoms:

- Backend logs show "connection timeout"
- Prisma errors: "Can't reach database"

Solutions:

```
# Verify DATABASE_URL in .env
cat .env | grep DATABASE_URL

# Check database is running
docker-compose ps database

# Verify database hostname matches service name
# Should be: database:1433 (NOT localhost:1433)

# Test database connectivity
docker-compose exec backend ping database

# Check database logs
docker-compose logs database
```

Issue 3: 502 Bad Gateway

Symptoms:

- Nginx returns 502 error
- Frontend/Backend not accessible

Solutions:

```
# Check if containers are running
docker-compose ps

# Verify port mappings
docker-compose ps | grep "0.0.0.0"

# Check Nginx configuration
sudo nginx -t

# Reload Nginx
sudo systemctl reload nginx

# Check Nginx error logs
sudo tail -f /var/log/nginx/error.log
```

Issue 4: SSL Certificate Issues

Symptoms:

- Browser shows "Not Secure"
- SSL certificate expired or invalid

Solutions:

```
# Check certificate status
sudo certbot certificates

# Renew certificate
sudo certbot renew

# Force renewal
sudo certbot renew --force-renewal

# Test auto-renewal
sudo certbot renew --dry-run
```

Issue 5: CI/CD Pipeline Fails

Symptoms:

- GitHub Actions workflow shows red X
- Deployment doesn't complete

Solutions:

```
# Check GitHub Actions logs
# Go to: https://github.com/user/repo/actions

# Verify GitHub Secrets are set correctly
# Settings → Secrets → Actions

# Test SSH connection locally
ssh -i ~/.ssh/github_actions_deploy deployer@YOUR_VPS_IP

# Verify Docker Hub login
docker login

# Check VPS disk space
df -h

# Clean up Docker resources
docker system prune -a --volumes -f
```

Issue 6: Environment Variables Not Loaded

Symptoms:

- Backend shows "undefined" for env variables
- CORS errors despite correct configuration

Solutions:

```
# Verify .env file on VPS
cat .env

# Restart containers to reload .env
docker-compose down && docker-compose up -d

# Check if backend reads .env
docker-compose exec backend printenv | grep DATABASE_URL

# Ensure .env is in same directory as docker-compose.yml
ls -la
```

Diagnostic Commands

```
# Full system status
docker-compose ps
docker-compose logs --tail=50
sudo systemctl status nginx
sudo systemctl status docker

# Network diagnostics
docker network ls
docker network inspect project-name_app-network

# Resource usage
docker stats --no-stream
df -h
free -m

# Port availability
sudo netstat -tulpn | grep -E ':(80|443|8080|8081|1433)'

# Check DNS resolution
nslookup yourdomain.com
dig yourdomain.com

# Test SSL
openssl s_client -connect yourdomain.com:443 -servername yourdomain.com
```

▯ Maintenance & Updates

Regular Maintenance Tasks

Daily/Automated

- Monitor container health
- Check log files for errors
- Verify SSL certificate auto-renewal
- Backup database (automated cron job)

Weekly

```
# Check disk usage
df -h
docker system df

# Review application logs
docker-compose logs --since 7d > weekly-logs.txt

# Update Docker images
docker-compose pull
docker-compose up -d

# Cleanup old images
docker image prune -a -f --filter "until=168h"
```

Monthly

```
# System updates
sudo apt update && sudo apt upgrade -y

# Docker updates
sudo apt install docker-ce docker-ce-cli containerd.io

# Review security advisories
docker scan <image-name>

# SSL certificate check
sudo certbot certificates

# Database maintenance
docker-compose exec database # run database optimization
```

Database Backup

Automated Backup Script:

```
# Create backup script
sudo nano /usr/local/bin/backup-database.sh
```

```
#!/bin/bash
# Database backup script

BACKUP_DIR="/var/backups/database"
DATE=$(date +%Y%m%d_%H%M%S)
CONTAINER_NAME="project-database"
DB_PASSWORD="YOUR_DB_PASSWORD"

mkdir -p $BACKUP_DIR

# For MS SQL Server
docker exec $CONTAINER_NAME /opt/mssql-tools18/bin/sqlcmd \
  -S localhost -U sa -P "$DB_PASSWORD" -C \
  -Q "BACKUP DATABASE app_db TO DISK = '/var/opt/mssql/backup_$DATE.bak'"

docker cp $CONTAINER_NAME:/var/opt/mssql/backup_$DATE.bak $BACKUP_DIR/

# Keep only last 7 days
find $BACKUP_DIR -name "*.bak" -mtime +7 -delete

echo "Backup completed: $DATE"
```

```
# Make executable
sudo chmod +x /usr/local/bin/backup-database.sh

# Add to crontab (daily at 2 AM)
sudo crontab -e

# Add line:
0 2 * * * /usr/local/bin/backup-database.sh >> /var/log/database-backup.log 2>&1
```

Deployment Updates

Process for deploying code changes:

1. **Local Development:**

   ```
   # Make code changes
   git add .
   git commit -m "feat: new feature"
   git push origin main
   ```

2. **Automatic Deployment:**
    - GitHub Actions triggers automatically
    - Tests run on both backend and frontend
    - Docker images built and pushed
    - VPS pulls new images and restarts
3. **Monitor Deployment:**
    - Watch GitHub Actions: `https://github.com/user/repo/actions`
    - Check VPS logs:

    ```
    ssh deployer@YOUR_VPS_IP
    cd /var/www/htdocs/project-name
    docker-compose logs -f
    ```
4. **Verify:**

    ```
    # Check container status
    docker-compose ps

    # Test API
    curl https://yourdomain.com/api/health

    # Check frontend
    # Open browser: https://yourdomain.com
    ```

Rollback Procedure

If deployment fails or introduces bugs:

```
# SSH to VPS
ssh deployer@YOUR_VPS_IP
cd /var/www/htdocs/project-name

# Check git log for previous working commit
git log --oneline -n 10

# Rollback to previous commit
git checkout <previous-commit-hash>

# Rebuild containers with old code
docker-compose down
docker-compose up -d --build

# Or use previous Docker images
docker-compose pull backend:backend-<previous-git-sha>
docker-compose pull frontend:frontend-<previous-git-sha>
docker-compose up -d

# Monitor
docker-compose logs -f
```

□ Security Best Practices

Security Checklist

- [ ] **Strong Passwords:** All passwords minimum 16 characters
- [ ] **SSH Key Authentication:** Disable password authentication
- [ ] **Firewall Enabled:** UFW configured with minimal open ports
- [ ] **SSL/TLS:** Always use HTTPS, redirect HTTP to HTTPS
- [ ] **Environment Variables:** Never commit secrets to git
- [ ] **Docker Security:** Run containers as non-root users
- [ ] **Regular Updates:** Keep all software up-to-date
- [ ] **Backup Strategy:** Automated daily backups
- [ ] **Log Monitoring:** Review logs for suspicious activity
- [ ] **Rate Limiting:** Implement API rate limiting

- [ ] **CORS Configuration:** Restrict to specific domains
- [ ] **SQL Injection Prevention:** Use ORM parameterized queries
- [ ] **XSS Protection:** Sanitize user inputs
- [ ] **Dependency Scanning:** Regular `npm audit`

SSH Hardening

```
# Edit SSH configuration
sudo nano /etc/ssh/sshd_config

# Recommended settings:
Port 2211                          # Change from default 22
PermitRootLogin no                 # Disable root login
PasswordAuthentication no          # Only allow key-based auth
PubkeyAuthentication yes
MaxAuthTries 3
MaxSessions 5
ClientAliveInterval 300
ClientAliveCountMax 2

# Restart SSH
sudo systemctl restart sshd

# Update firewall
sudo ufw delete allow 22/tcp
sudo ufw allow 2211/tcp
sudo ufw reload
```

Docker Security

```
# Run containers as non-root
# Add to Dockerfile:
USER node

# Scan images for vulnerabilities
docker scan wizoneit/project-name:backend
docker scan wizoneit/project-name:frontend

# Limit container resources
# Add to docker-compose.yml:
services:
  backend:
    deploy:
      resources:
        limits:
          cpus: '1.0'
          memory: 1G
        reservations:
          cpus: '0.5'
          memory: 512M
```

Application Security

```
# Backend security headers (Express.js)
const helmet = require('helmet');
app.use(helmet());

# Rate limiting
const rateLimit = require('express-rate-limit');
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per windowMs
});
app.use('/api/', limiter);

# CORS configuration
const cors = require('cors');
app.use(cors({
  origin: process.env.CORS_ORIGIN,
  credentials: true
}));

# Input validation
const { body, validationResult } = require('express-validator');
app.post('/api/users', [
  body('email').isEmail(),
  body('password').isLength({ min: 8 })
], (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }
  // Process request
});
```

Additional Resources

Documentation Links

- **Docker:** https://docs.docker.com
- **Docker Compose:** https://docs.docker.com/compose
- **Nginx:** https://nginx.org/en/docs
- **Let's Encrypt:** https://letsencrypt.org/docs
- **GitHub Actions:** https://docs.github.com/en/actions
- **Prisma:** https://www.prisma.io/docs
- **Node.js Best Practices:** https://github.com/goldbergyoni/nodebestpractices

Monitoring Tools

- **Portainer:** Docker container management UI

  ```
  docker volume create portainer_data
  docker run -d -p 9000:9000 --name=portainer --restart=always \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v portainer_data:/data portainer/portainer-ce
  ```

- **Uptime Kuma:** Self-hosted monitoring

  ```
  docker run -d --restart=always -p 3001:3001 \
    -v uptime-kuma:/app/data --name uptime-kuma \
    louislam/uptime-kuma:1
  ```

Useful Commands Reference

```
# Docker
docker ps                          # List running containers
docker ps -a                       # List all containers
docker logs <container>            # View logs
docker exec -it <container> sh     # Enter container shell
docker-compose up -d               # Start services
docker-compose down                # Stop services
docker-compose restart <service>   # Restart specific service
docker system prune -a             # Clean up everything

# Git
git status                         # Check status
git log --oneline                  # View commit history
git pull origin main               # Pull latest changes
git checkout <commit-hash>         # Checkout specific commit
git revert <commit-hash>           # Revert commit

# Nginx
sudo nginx -t                      # Test configuration
sudo systemctl reload nginx        # Reload config
sudo systemctl restart nginx       # Restart service
sudo tail -f /var/log/nginx/error.log  # View error logs

# SSL
sudo certbot certificates          # List certificates
sudo certbot renew                 # Renew certificates
sudo certbot delete --cert-name yourdomain.com  # Delete certificate

# System
htop                               # Process monitor
df -h                              # Disk usage
free -m                            # Memory usage
netstat -tulpn                     # Network ports
journalctl -u docker               # Docker service logs
```

□ Support & Troubleshooting

Getting Help

1. **Check logs first:** Most issues can be diagnosed from logs
2. **Search documentation:** Official docs usually have answers
3. **Community forums:** Stack Overflow, Docker Forums, GitHub Discussions
4. **GitHub Issues:** Report bugs or request features

Emergency Contacts

- **System Administrator:** [Your contact info]
- **DevOps Team:** [Team contact]
- **On-Call Support:** [Phone/Slack]

□ Deployment Completion Checklist

Print this checklist and check off each item:

Pre-Deployment

- ☐ Project structure follows standard format
- ☐ All Dockerfiles created and tested
- ☐ docker-compose.yml configured
- ☐ .gitignore includes .env files
- ☐ Environment variables documented in .env.example
- ☐ GitHub repository created

- [ ] Code pushed to GitHub

VPS Setup

- [ ] VPS provisioned and accessible
- [ ] Deployment user created
- [ ] Docker installed
- [ ] Docker Compose installed
- [ ] Nginx installed
- [ ] Certbot installed
- [ ] Firewall configured
- [ ] Project directory created

Domain & SSL

- [ ] Domain purchased
- [ ] DNS A records configured
- [ ] DNS propagation verified
- [ ] Nginx reverse proxy configured
- [ ] SSL certificate obtained
- [ ] HTTPS redirect working
- [ ] SSL auto-renewal tested

GitHub Configuration

- [ ] All secrets added to GitHub
- [ ] SSH key configured for deployment
- [ ] Docker Hub token created
- [ ] CI/CD workflow file created
- [ ] Workflow tested and passing

Database

- [ ] Database container running
- [ ] Migrations executed
- [ ] Seed data loaded (if applicable)
- [ ] Admin user created
- [ ] Backup strategy implemented

Application

- [ ] Backend container healthy
- [ ] Frontend container healthy
- [ ] Database container healthy
- [ ] API endpoints responding
- [ ] Frontend loads correctly
- [ ] Login functionality works
- [ ] No CORS errors
- [ ] SSL certificate valid

Post-Deployment

- [ ] All tests passing
- [ ] Documentation updated
- [ ] Team notified
- [ ] Monitoring configured
- [ ] Backup verified
- [ ] Rollback plan documented

� Revision History

Version	Date	Author	Changes
1.0	Jan 21, 2026	Wizone IT	Initial SOP created

Version	Date	Author	Changes

End of SOP Document

For questions or updates to this document, please contact the DevOps team or create an issue in the project repository.